On the Learning Capabilities of Recurrent Neural Networks: A Cryptographic Perspective

Shivin Srivastava
Dept. of Computer Science and Information Systems
Birla Institute of Technology and Sciences
Pilani, India
h2013073@pilani.bits-pilani.ac.in

Ashutosh Bhatia
Dept. of Computer Science and Information Systems
Birla Institute of Technology and Sciences
Pilani, India
ashutosh.bhatia@pilani.bits-pilani.ac.in

***Abstract*—It has been proven that Recurrent Neural Networks (RNNs) are Turing Complete, i.e. for any given computable function there exists a finite RNN to compute it. Consequently, researchers have trained Recurrent Neural Networks to learn simple functions like sorting, addition, compression and more recently, even classical cryptographic ciphers such as the Enigma. In this paper, we try to identify the characteristics of functions that make them easy or difficult for the RNN to learn. We look at functions from a cryptographic point of view by studying the ways in which the output depends on the input. We use cryptographic parameters (confusion and diffusion) for determining the strength of a cipher and quantify this dependence to show that a strong correlation exists between the learning capability of an RNN and the function's cryptographic parameters.**

***Index Terms*—Recurrent Neural Networks; Cryptographic Ciphers; Confusion Parameter; Diffusion Parameter;**

I. Introduction

Recently Deep Neural Networks, especially RNNs, have become quite popular for their state-of-the-art performance in many tasks like speech recognition, handwriting recognition/generation, image captioning etc. It has also been shown that they can learn simple algorithmic tasks like sorting and addition. Recently, it has been shown by Graydanus [1] that Long Short Term Memory (LSTM) [2] based models can even learn the decryption algorithm of classical cryptographic ciphers like the Vigenere, Autokey and more famously the Enigma cipher given the key and the ciphertext. This is a quite significant discovery as tasks like sorting, searching and addition are much easier to learn for humans as compared to learning the Enigma. The Vigenere cipher was also considered to be unbreakable for several centuries [3].

In one sense, the above breakthrough is not surprising since RNNs were proven to be Turing complete back in 1992 by by H. T. Siegelmann et. al. [4]. Given any computable function, there exists a finite RNN (with suitable weights) that can compute it. But this still does not shed any light on the practical aspects of learning a function. More specifically, we still do not know how fast our model will learn the function, how many training samples it will need to see, will it learn at all or not etc. All these questions clearly depend on the particular algorithm which we are interested to learn. So the real question is how to tell whether an RNN will be able to learn an algorithm? More clearly, "Can we quantify the learning capabilities of RNN on a given algorithmic task?"

In this paper, we try to answer the above question by training RNN based models to learn various algorithmic tasks like classical cryptographic ciphers and simple arithmetic tasks like long, addition, long multiplication and arithmetic modulus. We choose classical cryptographic functions as they are computationally easy to evaluate and at the same time they transform the input sequence in a non-trivial way.

We frame these algorithms as sequence-to-sequence translation tasks and use a single model with same topology and hyperparameters to learn them. We quantify the property of complexity of an algorithm by borrowing the concept of confusion and diffusion from the theory of cryptographic functions. We show that there exists a direct correlation between the confusion and especially the diffusion parameter of an algorithm and the learning capability of the RNN in-terms of test accuracy, saturation time and the number of iterations needed to achieve 95% accuracy (See section VI-D for an explanation of these terms).

Since we are training our model on various ciphers, we use the same model (Section V), parameters and problem definition (Section III) as described by [1].

II. Related Work

There have been many notable applications of Recurrent Neural Networks, in particular those which have a Long Short Term Memory (LSTM) to many interesting tasks such as Speech Recognition [5], Handwriting recognition [6], Image Captioning [7], Language Translation [8] etc. RNNs can find general solutions to algorithmic tasks. Zaremba and Sutskever [9] trained an LSTM to perform long addition of two 9-digit numbers with 99% accuracy. In [10], Graves et. al. introduced Neural Turing Machines (NTM) and compared them with LSTMs on simple tasks like reading, writing and repeated copying. In 2016, Graves et. al. [11] introduced, Differential Neural Computer (DNC) which is an extension of NTM combining attention mechanisms. It can solve problems like relational reasoning over graphs. However, none of the above works try to classify algorithms as easy and difficult from the point of view of RNN.

978-1-5386-9125-0/18/$31.00 ©2018 IEEE
DOI 10.1109/ICBK.2018.00029

Some work has been done on breaking ciphers using Artificial Intelligence algorithms. Spillman et. al. [12] use genetic algorithms to break a simple substitution cipher. Graydanus [1] successfully learned the decryption mechanism of the Enigma cipher using LSTMs. They also conduct the cryptanalysis of Vigenere and Autokey ciphers using LSTMs. Although they show that the Enigma cipher is much more difficult to learn as compared to Autokey cipher which is more difficult than the Vigenere cipher, they do not investigate further as to why it is so. In this paper, we try to answer this very question.

III. Problem Setup

We view all the algorithms/functions from a cryptographic perspective and use the general word 'cipher' for the same. The first input sequence to the cipher consists of the plaintext P concatenated with the key K. While the output sequence is called the ciphertext C. The encryption function is denoted by E. P, K, C are sequences consisting of characters belonging to the alphabet A^l(of length l). Our main objective is to train a neural network with parameters θ to make the approximation $\hat{E}_{Net}(K, P, \theta) \approx E(K, P)$ such that:-

$$\hat{\theta} = argmin_\theta \mathcal{L}(\hat{E}_{Net}(K, P, \theta) - E(K, P))$$

where $\mathcal{L}$ is the L_2 loss function. $E(K, P)$ is a one-hot encoded vector and $\hat{E}_{Net}$ is a real-valued softmax distribution over the same space.

The long addition, long multiplication and arithmetic modulo operation are also modeled as a cipher with the first input being called the plaintext while the second input takes the role of the key. The 'sort' cipher does not use a key. It sorts the string in alphabetic order.

The decryption process can be stated as $P = D(C, K)$, where C is the ciphertext (input), K is the key, P is the plaintext (output) and D is the decryption function. P, K, and C are sequences of symbols drawn from alphabet A^l (which has length l).

For addition cipher, the decryption function is arithmetic subtraction while integer division is the decryption function for multiplication. The modulo and the sort cipher have no decryption function.

A. Representing Ciphers

We borrow the representation methodology used in [1] as it works well in practice. A^{27} is chosen to be the uppercase Roman alphabet plus the null symbol, $'-'$. In addition to A^{27}, we also use the dictionary N^{11} which is the set of 10 Hindu-Arabic numerals plus the null symbol, $'-'$. Each symbol a_i in A^{27} or N^{10} is encoded as a one-hot vector. For computational purposes, we restrict the size of P to be 20 while the length of key K is varied from 1 to 6. We append the key phrase to the target sequence due to empirical benefit found by [1].

We consider 8 ciphers which use A^{26} as their dictionary. These are the Double Autokey, Double Vigenere, Autokey, Vigenere, Columnar Transposition, Simple Substitution, Affine cipher and the Sort cipher. These 8 ciphers are collectively called *cryptographic ciphers* in the rest of the paper. A sample input/output pair for the Vigenere cipher looks as following:-

input : `KEYV--TWOBIRDSITTINGONTREE`
output: `KEYV--EBNXTWCDNSPTSFKYYQAP`

The Double Vigenere and the Double Autokey ciphers employ two keys and the input/output pairs for these ciphers are as follows:-

input : `KA-KB-TWOBIRDSITTINGONTREE`
output: `KA-KB-PZKEEUZVOLPWEQCRJWNHA`

We also consider 3 ciphers which use N^{10} as their dictionary. These are the 20-digit long addition, long multiplication and the arithmetic modulo operation. These 3 are collectively called as *Arithmetic ciphers* in the rest of the paper.

As will be explained in section VI-B, for the `mult` cipher $|C| \leq |P| + |K|$ so we pad P with $|K|$ extra null characters ($'-'$) before hand only so that the length of plaintext and ciphertext is same for the LSTM model. Similarly, for the `add` cipher, we pad P with an extra $'-'$ character as $|C| \leq |P|+1$. So a typical input/output pair for the `add` cipher looks as following.

input : `1237657890-1234567891`
output: `-2472225781`

TABLE I: Diffusion and Confusion parameters

Cipher	Diffusion	Confusion
Double Autokey	0.158	0.32
Autokey	0.058	0.048
Affine	0.032	0.86
Columnar Transpose	0.032	0.36
Substitution	0.032	0.074
Vigenere	0.032	0.28
Double Vigenere	0.048	0.64
20-digit multiplication	0.559	0.457
20-digit modulo	0.213	0.412
20-digit addition	0.196	0.113
Sort	0.25	-

IV. Confusion and Diffusion parameters of different ciphers

In his seminal work on cryptography [13], Claude Shannon introduced the concept of confusion and diffusion methods to frustrate statistical attacks on the cipher. Generally, these parameters are defined for ciphers which work at bit level. An ideal cipher must have the avalanche effect property i.e. if a single bit changes in the plaintext then the output should change significantly (around half of the bits). Since we are working with classical ciphers which work at character level, instead of calculating the effect of a single bit change we study the effect of changing a single character in the plaintext and its resultant effect in the ciphertext. Now we describe in detail the method of confusion and diffusion and how we calculate them for the ciphers.

A. Confusion

Method of confusion is to make the relation between the simple statistics of the ciphertext C and the simple description of the key K a very complex one. Simply put, if the attacker is trying to solve for the key after gathering certain statistics

from the cipher text C (eg. frequency counts), then the method of confusion dictates that every statistic must depend on all parts of the key space so that it is not easy to solve for the key.

In our paper, we calculate the confusion parameter (χ) of a function E by generating a random pair of plaintext and key, (P, K) and finding the ciphertext $C = E(P, K)$. Then we choose a random position in K and exchange the character at that position with a different character from the dictionary, we get the modified key K'. The modified ciphertext C' is calculated as $C' = E(P, K')$. We use the Levenshtein distance (L) [14] to calculate the dissimilarity between C and C'. Thus we get $\chi(E) = \frac{L(C,C')}{|P|}$. This procedure is repeated for a large number of randomly generated (P, K) pairs to estimate the true value of the $\chi(E)$.

B. Diffusion

The method of diffusion is meant to dissipate the statistical structure of P which causes redundancy, into long range statistics i.e. into statistical structure involving a long combination of letters in the ciphertext. Generally this is achieved by making every block in C depend on many blocks in P. So even if some blocks have a statistical correlation with each other, since we are involving many other blocks of P, it is hoped that the structure is diffused.

We calculate the diffusion parameter (Δ) by generating a pair of key and plaintext, (P, K) and finding the ciphertext $C = E(P, K)$. The diffusion parameter is thus calculated as $\Delta(E) = \frac{L(P,C)}{|P|}$. This procedure is repeated for a large number of randomly generated (P, K) pairs to estimate the true value of $\Delta(E)$.

Table I, shows the Confusion and Diffusion parameters for various ciphers after averaging over 1000 different (P, K) pairs. We fix $|P| = 20$ and $1 \leq |K| \leq 6$. There are two exceptions to the key size though, for the Affine cipher $|K| = 2$ while for the Substitution cipher $|K| = 26$ (see section VI-B). All ciphers except the three arithmetic ciphers sample P and K from A^{26}. The arithmetic ciphers use N^{10} as dictionary.

V. Learning Model Architecture

The standard RNN cell maps an input sequence $\mathbf{x} = (x_1, ..., x_T)$ to an output vector sequence $\mathbf{y} = (y_1, ..., y_T)$ by using hidden states $\mathbf{h} = (h_1, ..., h_T)$ and iterating through the following equations from $t = 1$ to $t = T$:

$$h_t = H(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$
$$y_t = W_{hy}h_t + b_y$$

W denotes the weight matrices and b denotes the bias vectors. H is the hidden layer function and T is the length of input sequence.

H is frequently chosen to be the Long-Short-Term-Memory (LSTM) architecture which used built-in memory cells to store information. The memory vector c_t can be read, written to or reset at each time step. Thus the LSTM update takes the form:-

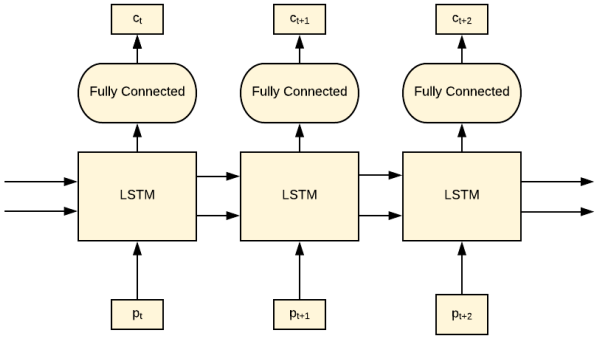

Fig. 1: RNN based model unrolled for three time steps

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \cdot c_{t-1}^l + i \cdot g$$
$$h_t^l = o \cdot tanh(c_t^l)$$

i,o and f are 3 gate vectors which decide whether the memory is updated, reset to zero or whether it is shown to the hidden vector respectively. The entire LSTM cell is differentiable allowing us to calculate its gradient function, and the three gating functions are helpful in reducing the problem of vanishing gradients ([15] and [16]).

We use a similar architecture as used in [1], where a single LSTM cell is capped with a fully connected softmax layer. In all experiments our LSTM network has 512 hidden layer nodes. It was observed that increasing the number of layers leads to very slow learning.

VI. Experimental Analysis

In this section, we verify the validity of our claims. We consider eleven different ciphers and train the LSTM model to learn the same. They include eight classical cryptographic ciphers and three arithmetic cipher as shown in Table I. The model of the Neural Network is as described in section V. All code for this project can be found on `https://github.com/shivin9/crypto-rnn`.

A. Test Setup

We perform all experiments on a Windows 10 workstation which has Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz processor with 32 GM RAM. We write our programs in Python using Google's TensorFlow library. The workstation has two Nvidia Quadro K620 GPUs which provide significant speedup during training.

The model is trained using the minibatch stochastic gradient descent algorithm with a constant learning rate of 5×10^{-4}. Every minibatch contains 64 training data samples. Every data

sample consists of a 6-character long key and a 20-length long plaintext. We use Xavier initialization to initialize the weights of the network. Since, we generate training samples on the fly, the model sees each training data sample only once. We take this approach due to two reasons:- 1) The model needs a large dataset in the training stage ($\approx 10^7$), so it is cheaper to generate the data on fly instead of storing and loading it into the memory repeatedly. 2) We want to train our model to learn the algorithm and not unrelated patterns between input and output. So, if it sees a (P, C) pair only once, then there is no chance of overfitting.

In all experiments, we report the test error by evaluating our model on 100 data samples and taking their average. The test accuracy is defined as the dot-product of one-hot representations of prediction and ground truth over all the characters in a sentence divided by the total number of characters. Accuracy between two words is defined as $Acc(y, \hat{y}) = y \cdot \hat{y} = \sum_{i=1}^{26} y_i \hat{y_i}$. It is calculated after every 500 minibatches and one training iteration consists of 100 minibatches. We train our model until it achieves $95\%+$ accuracy or trains for $150,000$ steps.

B. Ciphers

We briefly describe the different ciphers which we learn using the LSTM model.

1) Vigenere Cipher: Vigenere cipher is one of the most famous polyalphabetic substitution cipher. It shifts the plaintext characters by an amount decided by the key. This makes it harder to do a frequency analysis of the ciphertext. It can represented mathematically as follows :- $C[i] = A[(P[i] + K[i \ mod \ z]) \ mod \ A^l]$ where $z = |K|$ and A^l is the dictionary.

2) Autokey Cipher: Autokey cipher is a more sophisticated version of the Vigenere cipher where the key is used to only once. After the first $|K|$ message characters have been encrypted using the key, the plaintext is itself appended to the key and is used to encrypt further characters using substitution.

3) Affine Cipher: Affine cipher is a sophisticated form of Caesar cipher which just creates the substitution table in a more clever way by using a simple linear equation of the form $P[i] = (a * C[i] + b) mod A^l$ where A^l is the dictionary. The key K consists of only two characters generally represented as (a, b). It can be easily shown that a can only take 12 distinct values while b can take all 26 values.

4) Columnar Transposition cipher: Transposition ciphers change the positioning of characters in the plaintext instead of changing them directly. The plaintext is wrapped around a key after padding. Then the letters of the key are sorted along with the columns of plaintext. Ciphertext is basically obtained by reading the characters along the rows.

5) Substitution Cipher: Substitution cipher maintains a table which substitutes a plaintext character with a fixed character. The key K is a 26 characters long permutation of A with a character at position i being substituted with $K[i]$.

6) Double Vigenere Cipher: The Double Vigenere Cipher encrypts P twice with the Vigenere cipher using two different keys K_1 and K_2. It is actually equivalent to a single Vigenere cipher which has a key of length $lcm(|K_1|, |K_2|)$.

TABLE II: Learning parameters of the model for cryptographic ciphers

Cipher(E)	$\Delta(E)$	$\chi(E)$	**Test Accuracy**	**Sat. Time**	**95% Mark**
Sort	0.25	NA	48.5	945	NA
Double Autokey	0.158	0.32	16.7	0	NA
Autokey	0.087	0.048	100.0	295	290
Affine	0.048	0.86	98.94	230	170
Double Vigenere	0.048	0.64	98.94	2665	2340
Col. Trans.	0.048	0.36	64.66	250	NA
Vigenere	0.048	0.28	99.56	155	125
Simple Substitution	0.047	0.074	95.66	1625	1635

TABLE III: Learning parameters of the model for arithmetic ciphers

Cipher(E)	$\Delta(E)$	$\chi(E)$	**Test Accuracy**	**Sat. Time**	**95% Mark**
Mult	0.559	0.457	67.99	945	NA
Mod	0.213	0.412	85.14	5	NA
Add	0.196	0.113	95.81	165	175

7) Double Autokey Cipher: The Double Autokey Cipher encrypts P twice using the Autokey cipher(E). It also employs two keys K_1 and K_2. In our paper we bound the length of both the keys by 3.

8) Arithmetic Ciphers: The Arithmetic ciphers sample P and K from N^{10} and apply the appropriate arithmetic operation ie. $+$, $*$ or $\%$. The `mult` cipher converts P and K into integers and then multiplies them normally to get the ciphertext. For brevity we show it as $C = P \times K$. In a similar way the operators are also applied.

Note that unlike the cryptographic ciphers the Arithmetic ciphers change the length of the ciphertext. In case of `mult`, $|C| \leq |P| + |K|$. In case of `add`, $|C| \leq |P| + 1$ and in case of `mod`, $|C| \leq |K|$. Similar to other ciphers we fix $|P| = 20$ and $|K| \leq 6$.

9) Sort Cipher: The `sort` cipher simply sorts P in alphabetical order. There is no key involved.

C. Datasets

We choose to generate training data samples on the fly as the execution time of $E(P, K)$ is very low for all ciphers. This reduces the chances of overfitting as it's highly unlikely that a single data point is seen more than once by the model. We used the famous Python package, `PyCipher` [17] for implementations of the Affine, Columnar Transposition and the Simple Substitution cipher. We implemented our own versions of the Autokey, Vigenere, Double Autokey, Double Vigenere, Arithmetic and Sort cipher.

The symbols of P and K are sampled randomly (with uniform probability) from A^{26}, the Roman Alphabet. Choosing characters randomly ensures that the model learns the cipher function and not the statistical distribution of English language or the common n-grams.

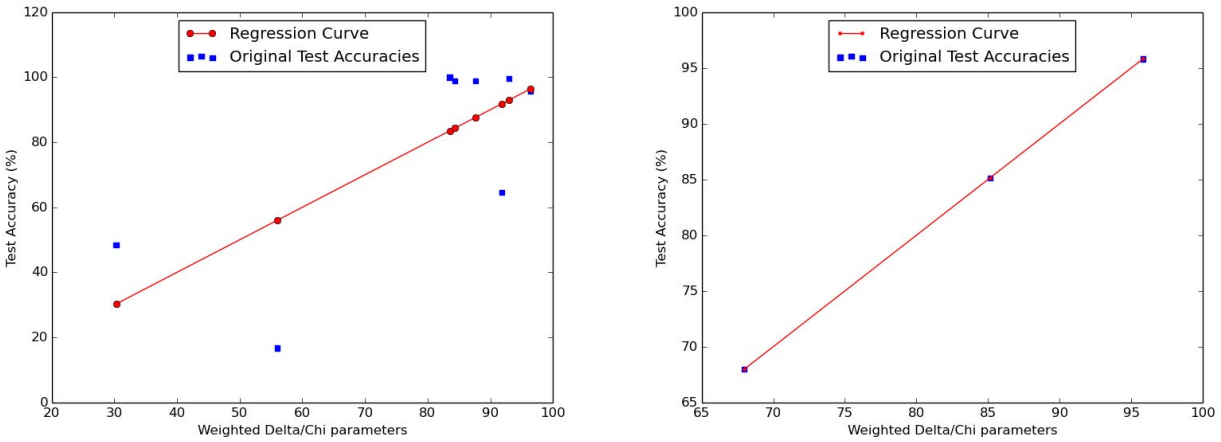

Fig. 2: Linear Regression Fitting of Test Accuracy as a function of Δ and χ parameters for (a) cryptographic ciphers; (b) arithmetic ciphers

D. Discussion

In this section we analyze the model on the following learning parameters:- final **Test Accuracy**, **Saturation Time** and 95%**-mark** while learning different ciphers. Saturation Time is defined as number of iterations needed to achieve an accuracy within 1% of maximum. 95%-mark denotes the number of iterations needed to reach 95% accuracy. The final test accuracy helps us judge the overall performance of the model while the Saturation Time helps us gauge the speed with which the model is learning the function and also the point at which its performance stabilizes. We don't use the maximum accuracy for this measure as the model may achieve maximum performance arbitrarily long after stabilizing.

The results are summarized in tables II and III. We have separated the results of Arithmetic and Cryptographic ciphers as they are learned for different dictionaries of different sizes. Both the tables are sorted by decreasing values of diffusion parameters of different ciphers.

In Table III, we can see a clear gradation in the learning parameters as a function of $\Delta(E)$ and $\chi(E)$. As $\Delta(E)$ increases, the Accuracy of the model decreases. The `Add` cipher is learned much better than the `Mult` and `Mod` cipher. The saturation time highlights a different phenomenon though, the model saturates quickly while learning the `Mod` cipher and it achieves a decent accuracy also. The model takes much more time to learn the `Mult` cipher and in the end its not able to learn it fully also. Thus, we can conclusively say that `Mult` cipher is tougher to learn as compared to the `Mod` and `Add` ciphers.

Table II on the other hand shows more interesting results. The `Sort` cipher and the Double Autokey cipher clearly follow the Δ/χ hypothesis when compared to other ciphers, although they do not follow it amongst themselves. The saturation times of both ciphers tells that although the model tried to learn the `sort` cipher it couldn't get much far even after a long effort. But the Double Autokey cipher remains a mystery for our model as it saturates in the very beginning only. Except for the Columnar transposition cipher, the model learns all other ciphers almost fully.

Simple Substitution cipher appears to not obey this rule, but we suspect that this is probably because it's key is the whole permutation table which is 26 characters long. For other ciphers, their keys are at most 6 characters long.

The only real outlier to the Δ/χ hypothesis is the Columnar Transposition cipher whose cryptographic parameters match the Vigenere cipher but is actually much difficult to learn in practice. We suspect that they are some other function parameters, unknown to us, which make the Columnar Transposition cipher difficult to learn. Another observation is that the model finds composition ciphers (Double Vigenere and Double Autokey) more difficult to learn as compared to the single ciphers even though they have similar cryptographic parameters. This suggests that the model finds difficulty in learning a composition of functions rather than a stand-alone function even though the key lengths for both are same.

The confusion parameter does not play a very major role in increasing the learning complexity of the cryptographic ciphers. But it does seem to act as the tie-breaker in cases where two algorithms match in their diffusion parameters. For example consider the Double Vigenere and the Vigenere cipher. The diffusion parameter for both is 0.048 but their confusion-parameters vary considerably. Consequently it is observed that the Double Vigenere cipher is much harder to learn as compared to the Vigenere cipher. Similarly the Affine and Vigenere cipher also follow this rule. Although, the model learns both Vigenere and Autokey ciphers fully, it learns the former much more quickly than the later as supported by our hypothesis.

To further strengthen the validity of the Δ/χ hypothesis, Figure 3 gives the heat map representation of table II which shows a pairwise comparison of the learning complexities of the cryptographic ciphers. If the predictions match the results obtained in the experiments, we color the box green, otherwise red. Cipher E_1 is termed more complex than cipher E_2 if the final accuracy of the LSTM model on E_1 is significantly less $(> 5\%)$ than that on E_2. In case of tie, we compare using the saturation time.

To determine the weight of Δ and χ parameters respectively on the final accuracy achieved by the model, we fit the data in table II using a linear regression model. The final equation we get is $Accuracy = -330.9 \times \Delta + -14.84 \times \chi + 113$. So the Δ parameter is around 22 times more significant than the χ parameter in deciding the learning complexity of the cryptographic ciphers. Figure 2(a) shows that the test accuracy does have a direct relationship (weakly linear with an R^2 value of 0.58) with the confusion and diffusion parameters of a cipher.

Similarly for table III, we get the following dependence of Test Accuracy on Δ and χ parameters:- $Accuracy = -45.26 \times \Delta + -33.11 \times \chi + 108.4$. The confusion parameter of Arithmetic ciphers is almost equally significant in determining its learning complexity. Figure 2(b) further strengthens the hypothesis that the Test Accuracy is a linear function of the cryptographic parameters of the arithmetic ciphers. Of course we will need more samples to strengthen this fact.

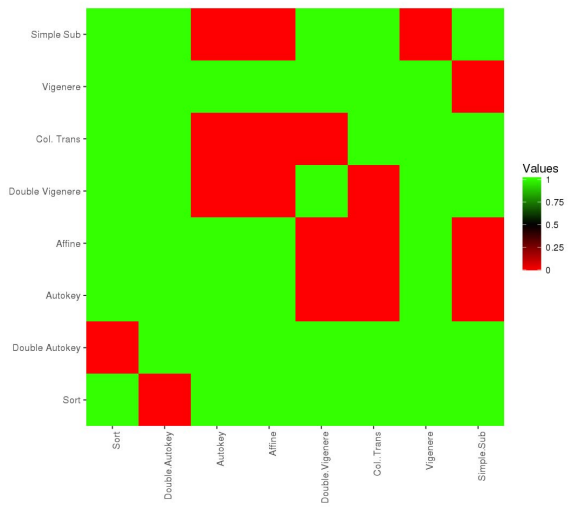

Fig. 3: Heat Map making a pairwise comparison between algorithms

VII. Conclusions and Future Work

In this work, we explore the relationship between a function's cryptographic (confusion and diffusion) parameters, and the capability of an RNN model to learn it when trained with valid input/output pairs. The experiments strongly suggest that confusion and diffusion parameters certainly reduce the learning capabilities of the 1-layer RNN model. However, there are some exceptions to our hypothesis, we thus suspect there there are certain other properties possessed by sequence-to-sequence functions which also restrict the RNN's generalization powers. In our view this work is just the beginning of what can become a fruitful technique for shedding light on the black-box nature of deep neural networks. Future works can focus on finding out more parameters inherent to functions which influence the learning capabilities of RNNs. The validity of the Δ/χ hypothesis can also be tested on different network architectures.

VIII. Acknowledgements

We are grateful to the anonymous reviewers for their helpful comments.

References

[1] S. Greydanus, "Learning the enigma with recurrent neural networks," *arXiv preprint arXiv:1708.07576*, 2017.
[2] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.
[3] "A new cipher code," *Scientific American v.83 suppl. 1917*, pp. 61–62, 1917.
[4] H. T. Siegelmann and E. D. Sontag, "On the computational power of neural nets," *Journal of computer and system sciences*, vol. 50, no. 1, pp. 132–150, 1995.
[5] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013, pp. 6645–6649.
[6] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.
[7] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3128–3137.
[8] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.
[9] W. Zaremba and I. Sutskever, "Learning to execute," *arXiv preprint arXiv:1410.4615*, 2014.
[10] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.
[11] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, p. 471, 2016.
[12] R. Spillman, M. Janssen, B. Nelson, and M. Kepner, "Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers," *Cryptologia*, vol. 17, no. 1, pp. 31–44, 1993.
[13] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.
[14] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8, 1966, pp. 707–710.
[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[16] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.
[17] J. Lyon, "Pycipher," 2014. [Online]. Available: https://github.com/jameslyons/pycipher